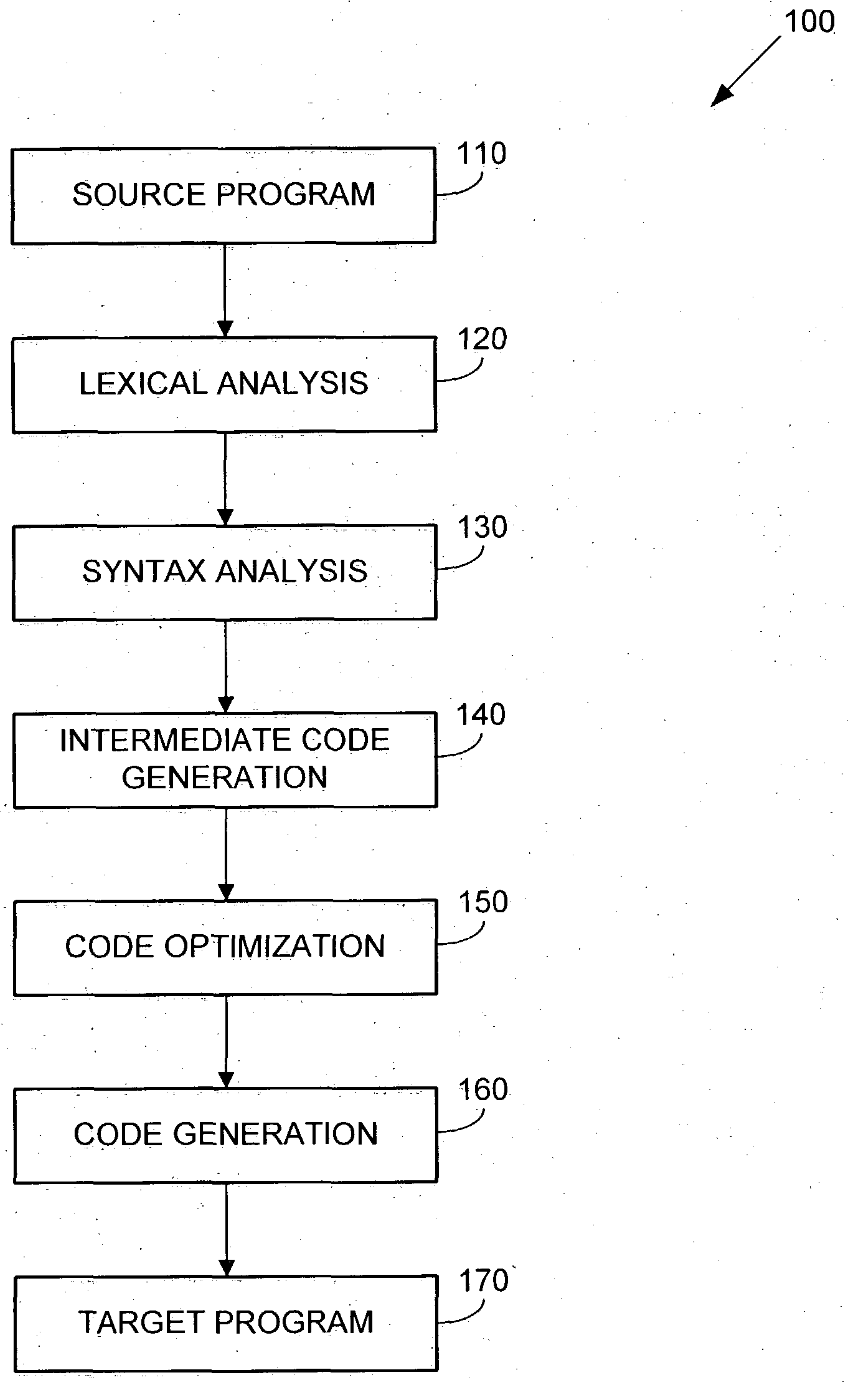
FIG. 1


100
SOURCE PROGRAM
110
LEXICAL ANALYSIS
120
SYNTAX ANALYSIS
130
INTERMEDIATE CODE GENERATION
140
CODE OPTIMIZATION
150
CODE GENERATION
160
TARGET PROGRAM
170

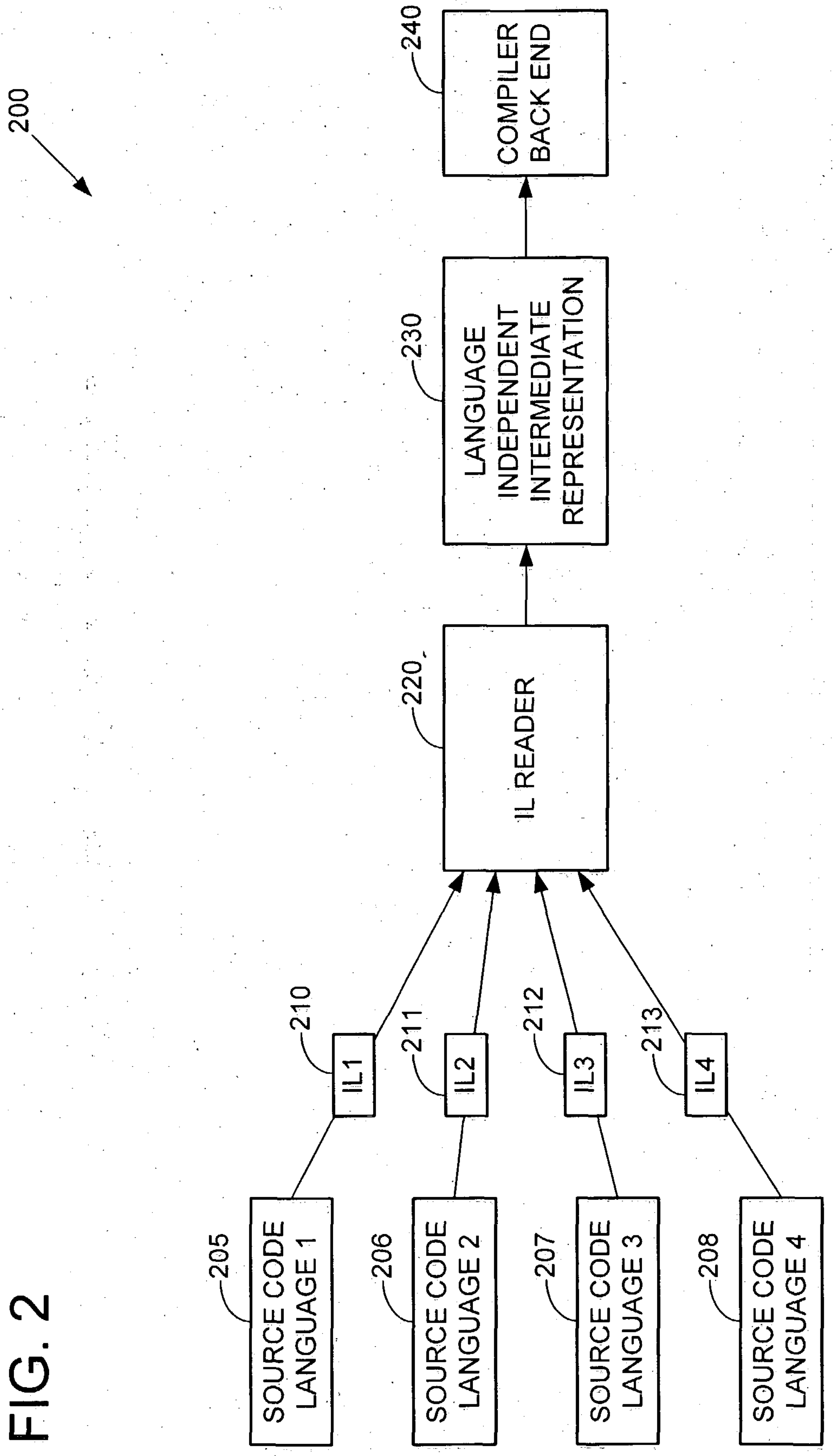

200
205 SOURCE CODE LANGUAGE 1
206 SOURCE CODE LANGUAGE 2
207 SOURCE CODE LANGUAGE 3
208 SOURCE CODE LANGUAGE 4
210 IL1
211 IL2
212 IL3
213 IL4
220 IL READER
230 LANGUAGE INDEPENDENT INTERMEDIATE REPRESENTATION
240 COMPILER BACK END

FIG. 2

FIG. 3A

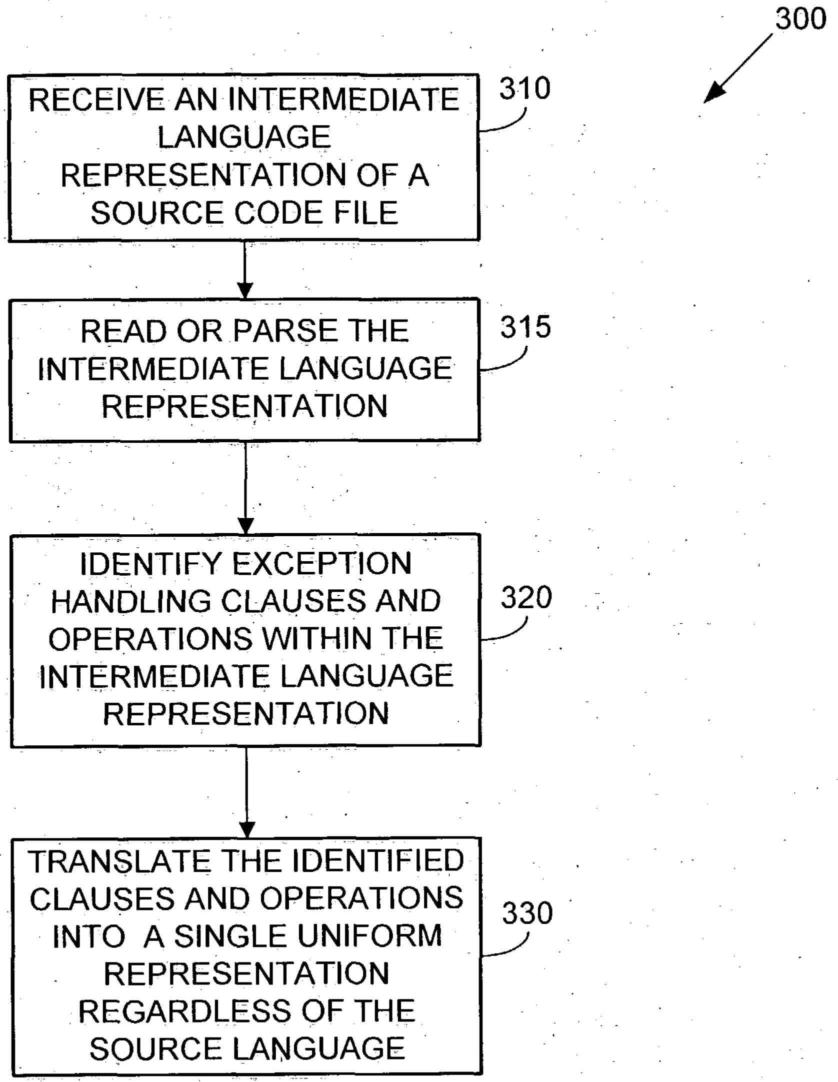

300
RECEIVE AN INTERMEDIATE LANGUAGE REPRESENTATION OF A SOURCE CODE FILE
310
READ OR PARSE THE INTERMEDIATE LANGUAGE REPRESENTATION
315
IDENTIFY EXCEPTION HANDLING CLAUSES AND OPERATIONS WITHIN THE INTERMEDIATE LANGUAGE REPRESENTATION
320
TRANSLATE THE IDENTIFIED CLAUSES AND OPERATIONS INTO A SINGLE UNIFORM REPRESENTATION REGARDLESS OF THE SOURCE LANGUAGE
330

FIG. 3B

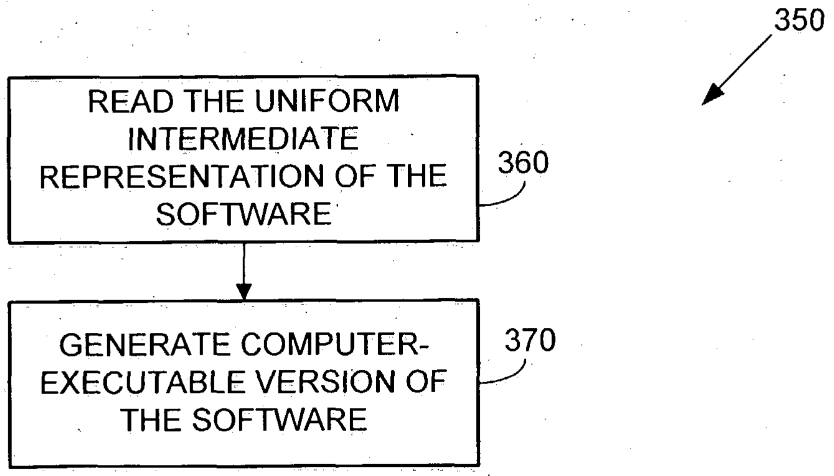

350
READ THE UNIFORM INTERMEDIATE REPRESENTATION OF THE SOFTWARE
360
GENERATE COMPUTER-EXECUTABLE VERSION OF THE SOFTWARE
370

FIG. 4

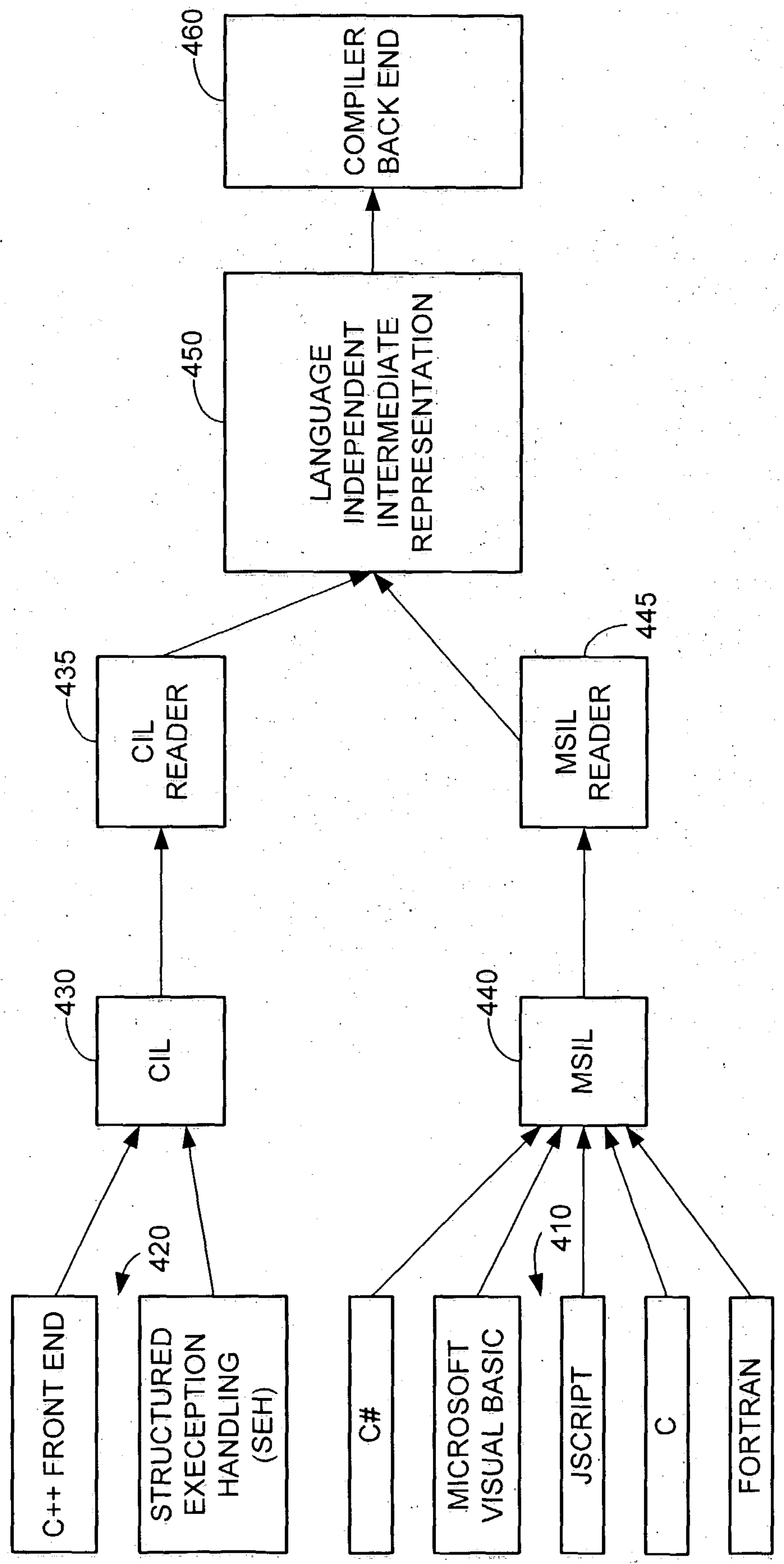
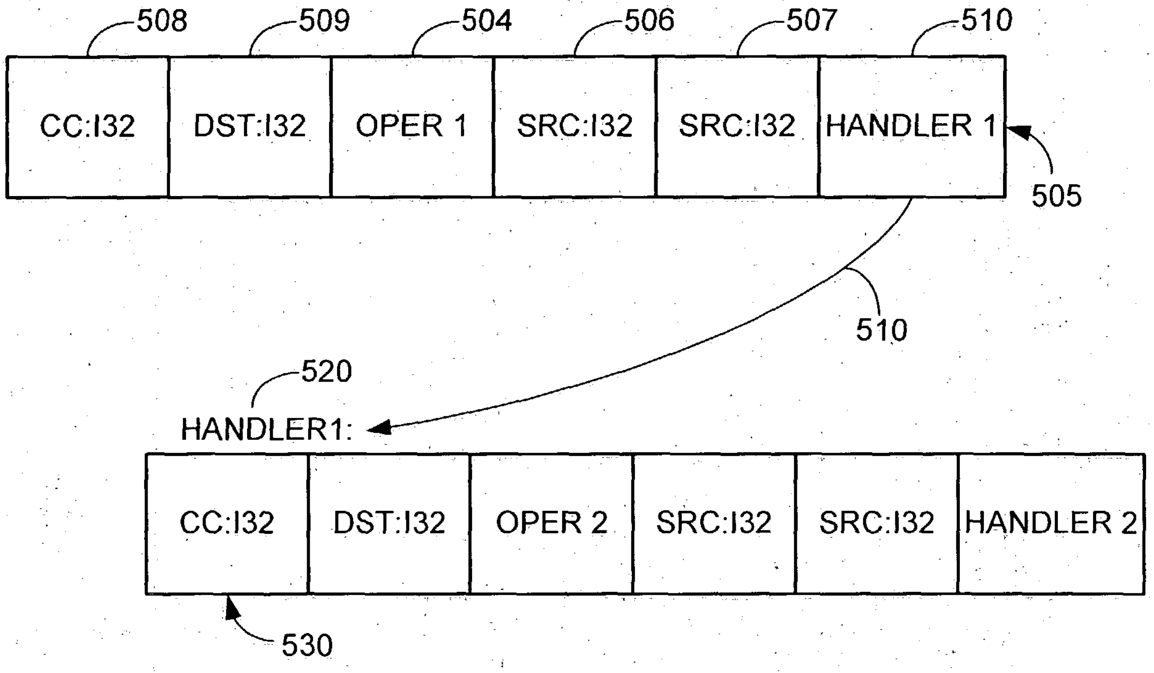

C++ FRONT END
STRUCTURED EXECEPTION HANDLING (SEH)
420
C#
MICROSOFT VISUAL BASIC
JSCRIPT
C
FORTRAN
410
CIL
430
MSIL
440
CIL READER
435
MSIL READER
445
LANGUAGE INDEPENDENT INTERMEDIATE REPRESENTATION
450
COMPILER BACK END
460

FIG. 5



508
509
504
506
507
510
CC:I32
DST:I32
OPER 1
SRC:I32
SRC:I32
HANDLER 1
505
510
520
HANDLER1:
CC:I32
DST:I32
OPER 2
SRC:I32
SRC:I32
HANDLER 2
530

FIG. 6

```
void foo(int a, int b, int c, int d)
{
  x = a div b;
  x = c div d;
}
```

FIG. 7

```
    a.int32, b.int32, c.int32 d.int32 = ENTER foo
    x.int32 = DIV a.int32, b.int32; $HANDLER  \
                                               }-710
    x.int32 = DIV c.int32, d.int32; $HANDLER  /
    EXIT
$HANDLER:
    UNWIND
    EXIT
```

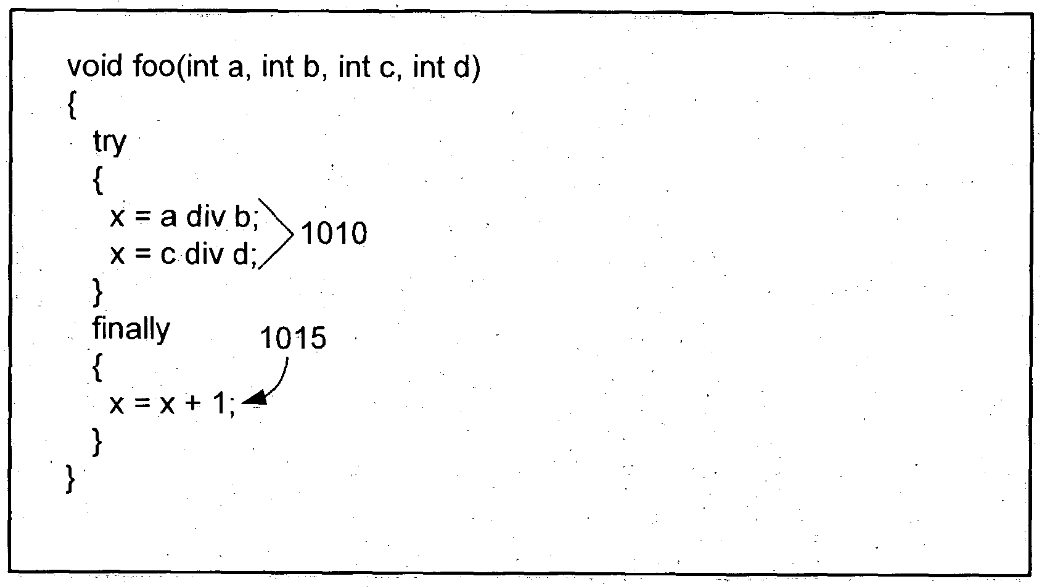
FIG. 8

```
void foo(int a, int b, int c, int d)
{
  try
  {                    810
    x = a + b;
    x = x + c * d;
  }
  finally          815
  {
    x = x + 1;
  }
}
```

FIG. 9

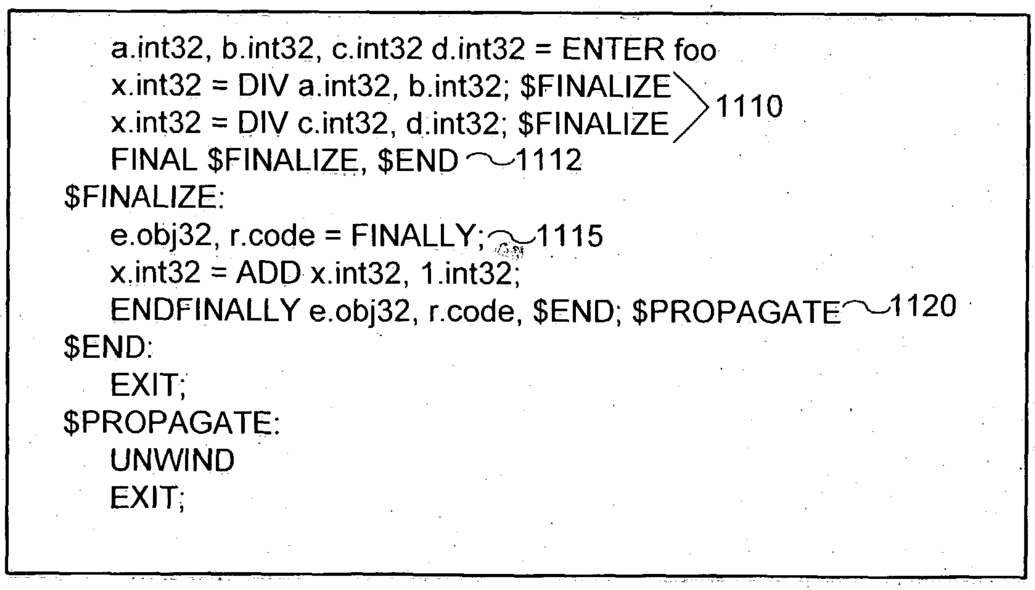
```
    a.int32, b.int32, c.int32 d.int32 = ENTER foo
    x.int32 = ADD a.int32, b.int32;        910
    t.int32 = MUL c.int32, d.int32;
    x.int32 = ADD x.int32, t.int32;
    FINAL $FINALIZE, $END          915
$FINALIZE:                          920
    e.obj32, r.code = FINALLY;
    x.int32 = ADD x.int32, 1.int32;
    ENDFINALLY e.obj32, r.code, $END;
$END:
    EXIT;
```

FIG. 10



```
void foo(int a, int b, int c, int d)
{
  try
  {
    x = a div b;
    x = c div d;
  }
  finally
  {
    x = x + 1;
  }
}
```

FIG. 11



```
    a.int32, b.int32, c.int32 d.int32 = ENTER foo
    x.int32 = DIV a.int32, b.int32; $FINALIZE
    x.int32 = DIV c.int32, d.int32; $FINALIZE
    FINAL $FINALIZE, $END
$FINALIZE:
    e.obj32, r.code = FINALLY;
    x.int32 = ADD x.int32, 1.int32;
    ENDFINALLY e.obj32, r.code, $END; $PROPAGATE
$END:
    EXIT;
$PROPAGATE:
    UNWIND
    EXIT;
```

1110
1112
1115
1120

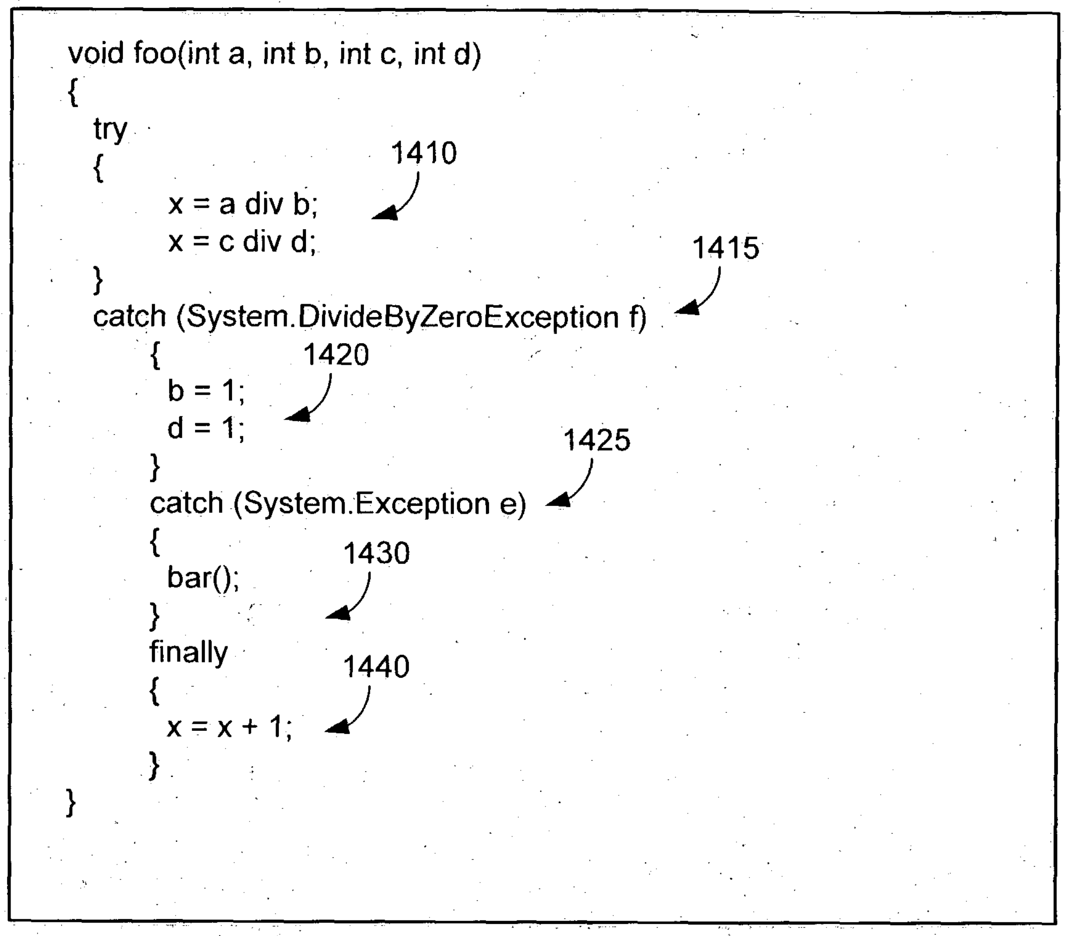
FIG. 12

```
1240
void foo(int a, int b, int c, int d)
{
 try                                      1210
{
x = a div b;
x = c div d;
}
catch (System.DivideByZeroException f) ~1215
{
b = 1;                                   1220
d = 1;
}
catch (System.Exception e) ~1225
{
bar();                                   1231
}
}
```

FIG. 13

```
    a.int32, b.int32, c.int32 d.int32 = ENTER foo
    x.int32 = a.int32 DIV b.int32; $HANDLER1 \
                                              > 1310
    x.int32 = c.int32 DIV d.int32; $HANDLER1 /
    GOTO $END$
$HANDLER1:                                        1315
    F.DivideByZeroException = TYPEFILTER $CATCH1,
HANDLER2;
$CATCH1:                                  1320
    b.int32 = ASSIGN 1.int32;
    d.int32 = ASSIGN 1.int32;
    GOTO $END
$HANDLER2:                                        1325
    E.Exception = TYPEFILTER $CATCH2, $PROPAGATE;
$CATCH2:                              1330
    CALL bar(); $PROPAGATE
    GOTO $END
$PROPAGATE:
    UNWIND ~1335
    EXIT;
```

FIG. 14



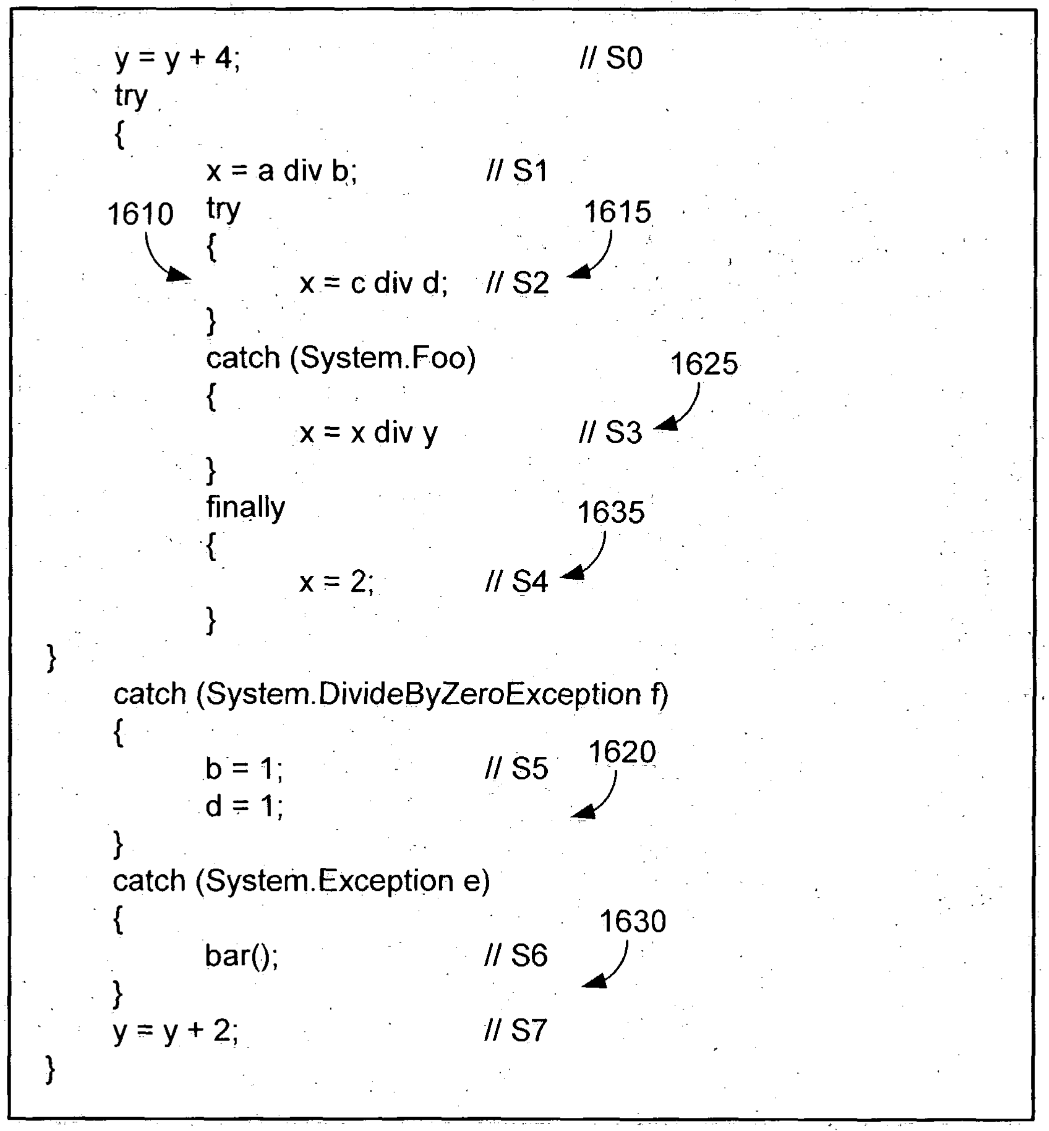
```
void foo(int a, int b, int c, int d)
{
  try
  {
        x = a div b;
        x = c div d;
  }
  catch (System.DivideByZeroException f)
      {
        b = 1;
        d = 1;
      }
      catch (System.Exception e)
      {
        bar();
      }
      finally
      {
        x = x + 1;
      }
}
```

1410
1415
1420
1425
1430
1440

FIG. 15

```
a.int32, b.int32, c.int32 d.int32 = ENTER foo
    x.int32 = a.int32 DIV b.int32; $HANDLER1 \
                                              > 1510
    x.int32 = c.int32 DIV d.int32; $HANDLER1 /
    FINAL $FINALIZE, $END
$HANDLER1:
    F.DivideByZeroException = TYPEFILTER $CATCH1, ~1515
$HANDLER2;
$CATCH1:                                        1520
    b.int32 = ASSIGN 1.int32;
    d.int32 = ASSIGN 1.int32;
    FINAL $FINALIZE, $END; ~1521
$HANDLER2:
    E.Exception = TYPEFILTER $CATCH2, $FINALIZE; ~1525
$CATCH2:                                        1530
    CALL bar(); $FINALIZE
    FINAL $FINALIZE, $END ~1531
$FINALIZE:                                      1540
    e.obj32, r.code = FINALLY;
    x.int32 = ADD x.int32, 1.int32;
    ENDFINALLY e.obj32, r.code, $END; $PROPAGATE
$PROPAGATE:  1550
    UNWIND
    EXIT;     1560
$END:
    EXIT;
```

FIG. 16



```
    y = y + 4;                          // S0
    try
    {
            x = a div b;            // S1
            try
            {
                    x = c div d;    // S2
            }
            catch (System.Foo)
            {
                    x = x div y         // S3
            }
            finally
            {
                    x = 2;          // S4
            }
}
    catch (System.DivideByZeroException f)
    {
            b = 1;                  // S5
            d = 1;
    }
    catch (System.Exception e)
    {
            bar();                  // S6
    }
    y = y + 2;                      // S7
}
```

1610
1615
1625
1635
1620
1630

FIG. 17

```
    y.int32 = ADD y.int32, 1.int32                                  // S0
    x.int32 = a.int32 DIV b.int32; $HANDLER1 ~1705                  // S1
    x.int32 = c.int32 DIV d.int32; $HANDLER3 ~1706                  // S2
    FINAL $FINALIZE, $S7;
$FINALIZE:                       1725
    e0, r0 = FINALLY
    x.int32 = ASSIGN 2.int32;                                       // S4
    ENDFINALLY e0, r0, $S7; $HANDLER1 ~1726
$HANDLER3:
    e1 = TYPEFILTER $CATCH3, $FINALIZE
$CATCH3:
    x.int32 = DIV x.int32, y.int32; $FINALIZE      1720             // S3
    FINAL $FINALIZE, $S7;   1710
$HANDLER1:
    e2 = TYPEFILTER $CATCH1, $HANDLER2;
$CATCH1:
    b.int32 = ASSIGN 1.int32;                                       // S5
    d.int32 = ASSIGN 1.int32;
    GOTO $S7;          1715
$HANDLER2:
    e3 = TYPEFILTER $CATCH2, $PROPAGATE;
$CATCH2:
    CALL bar(); $PROPAGATE;                                         // S6
    GOTO $S7
$PROPAGATE:
    UNWIND
    EXIT;
$S7:
    y.int32 = ADD y.int32, 1.int32                                  // S7
    GOTO $END;
$END:
    EXIT;
```

FIG. 18

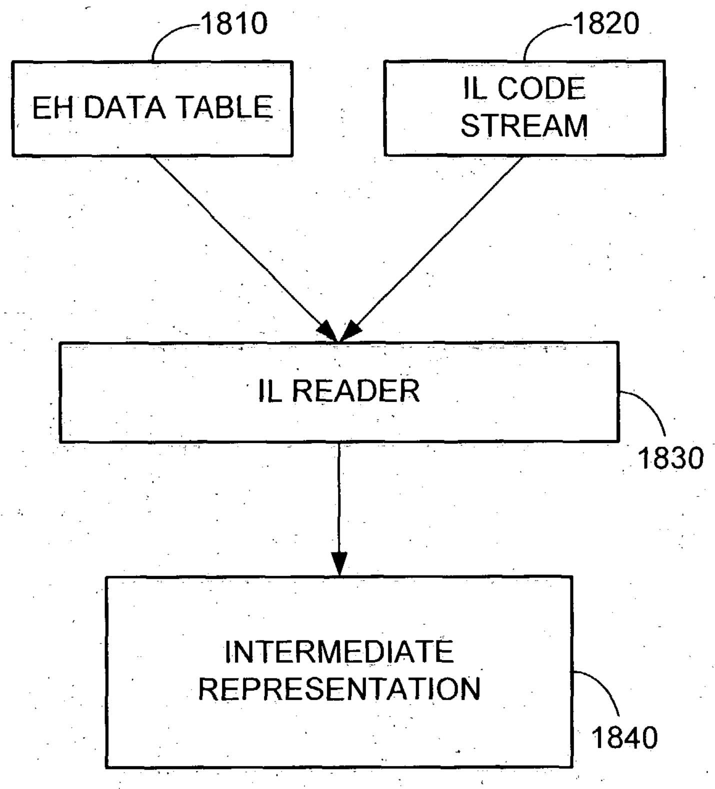

1810
EH DATA TABLE
1820
IL CODE STREAM
IL READER
1830
INTERMEDIATE REPRESENTATION
1840

FIG. 19A

1915 1910 1920

ENTRY	INFO TAG	PROTECTED BLOCK	DESTINATION/HANDLER BLOCK
1	TRY CATCH	3-7	8-12
2	TRY CATCH	3-7	13-16

FIG. 19B

DESTINATION/HANDLER BLOCK OFFSET	LABEL
8-12	HANDLER 1
13-16	HANDLER 2

FIG. 19C

PROTECTED BLOCK OFFSET	DESTINATION/HANDLER BLOCK OFFSET	LABEL
3-7	8-12	HANDLER 1
3-7	13-16	HANDLER 2

FIG. 20

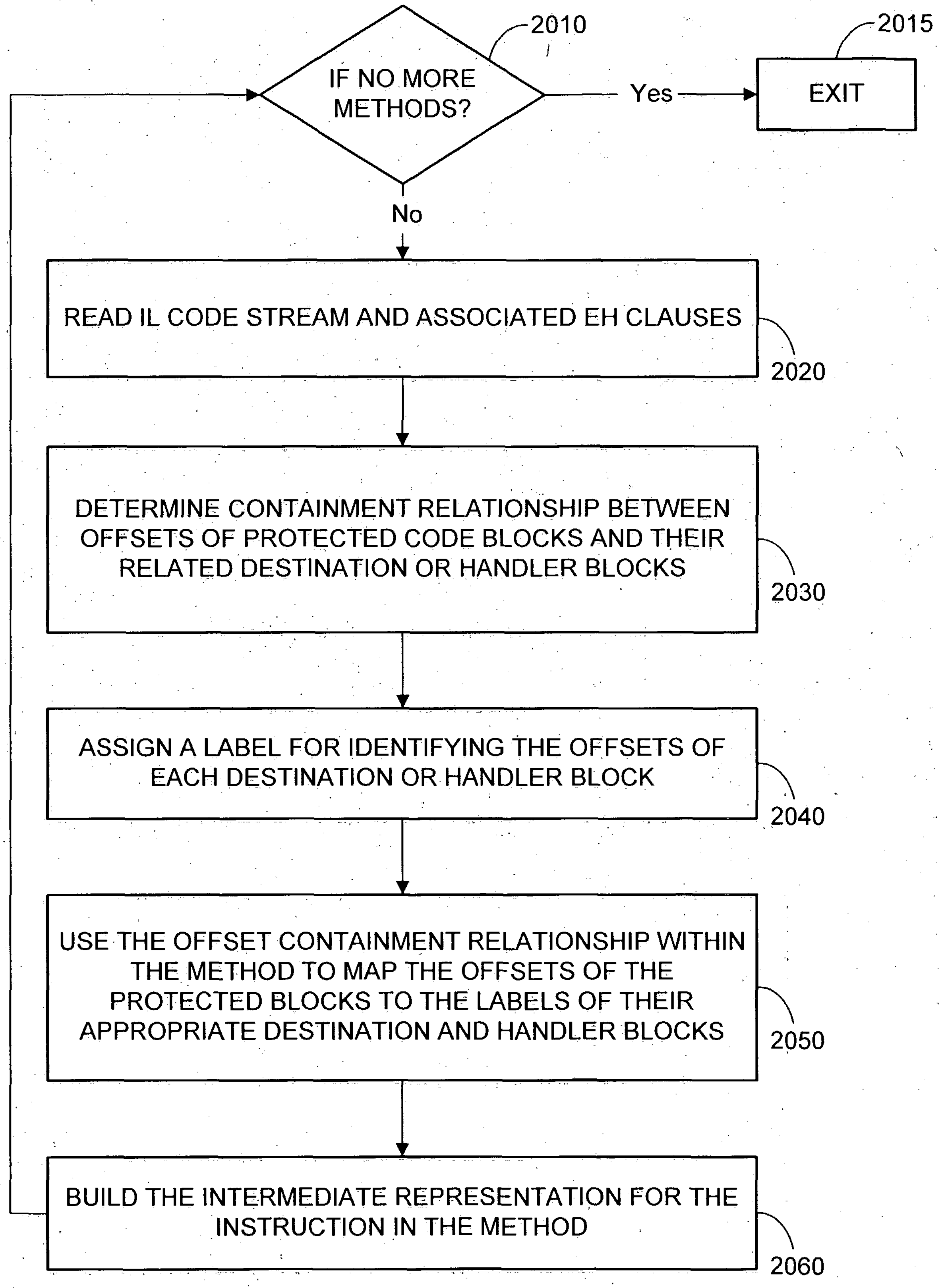

2010
IF NO MORE METHODS?
Yes
2015
EXIT
No
READ IL CODE STREAM AND ASSOCIATED EH CLAUSES
2020
DETERMINE CONTAINMENT RELATIONSHIP BETWEEN OFFSETS OF PROTECTED CODE BLOCKS AND THEIR RELATED DESTINATION OR HANDLER BLOCKS
2030
ASSIGN A LABEL FOR IDENTIFYING THE OFFSETS OF EACH DESTINATION OR HANDLER BLOCK
2040
USE THE OFFSET CONTAINMENT RELATIONSHIP WITHIN THE METHOD TO MAP THE OFFSETS OF THE PROTECTED BLOCKS TO THE LABELS OF THEIR APPROPRIATE DESTINATION AND HANDLER BLOCKS
2050
BUILD THE INTERMEDIATE REPRESENTATION FOR THE INSTRUCTION IN THE METHOD
2060

FIG. 21

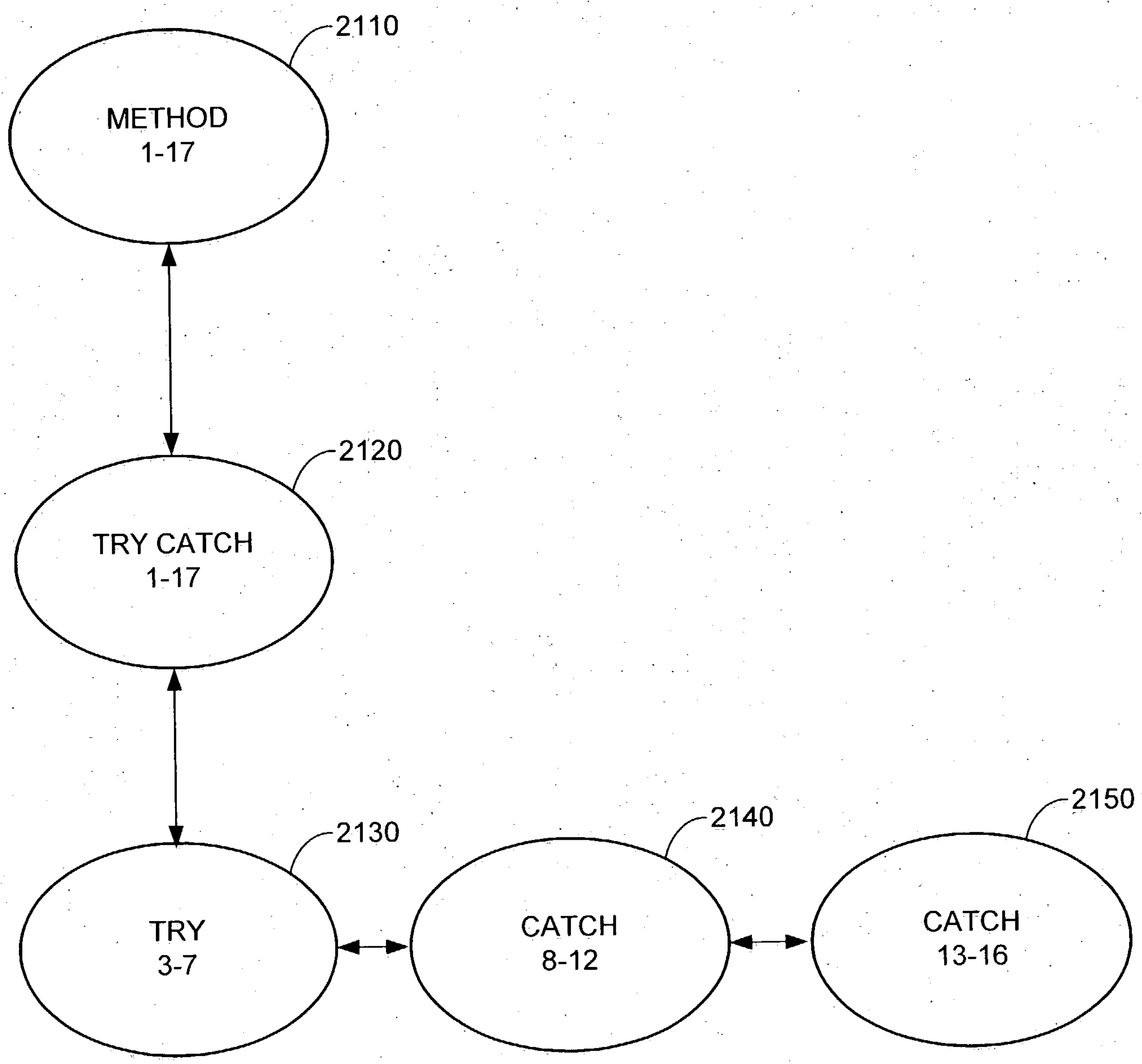

2110
METHOD 1-17
2120
TRY CATCH 1-17
2130
TRY 3-7
2140
CATCH 8-12
2150
CATCH 13-16

FIG. 22

```
void proc()
{
  class1 obj1; // S1
  obj1.foo();  // S2
  class2 obj2; // S3
  obj2.bar();  // S4
}
```

FIG. 23

```
void proc()
{                          2310
    ctor1(&obj1); <-------'
    try
    {
      obj1.foo();
      ctor2(&obj2);
           try
           {                          2330
                  obj2.bar(); <------'
           }
           finally
           {                              2340
                  dtor2(&obj2); <--------'
           }
    }
    finally
    {                            2320
           dtor1(&obj1) <-------'
    }
}
```

FIG. 24

```
ENTER proc
CALL class1, &_obj1   $PROPAGATE
CALL foo, &_obj1 $DTOR1
CALL class1,&_obj2 $DTOR1
CALL bar, &_obj2 $DTOR2
FINAL $DTOR2, $NEXT; ~2410
$NEXT:
    FINAL $DTOR1, $END ~2420
$DTOR2:
    e, r = FINALLY
    CALL DTOR2(&obj2); $DTOR1 ~2435          2430
    ENDFINALLY e, r, $DTOR1, $NEXT  <-
$DTOR1:
    e2, r2 = FINALLY                           2440
    CALL DTOR1(&obj1); $PROPAGATE ~2445
    ENDFINALLY e1, r2, $PROPAGATE, $END  <-
$PROPAGATE:
    UNWIND
    EXIT;
$END:
    EXIT;
```

FIG. 25

```
void proc(int x)
{
  foo(x ? obj1(x) : obj2(x+1));
}
```

FIG. 26

```
void proc()
{
    try
    {
     t1 = x ? ctor(&obj1,x) : NULL;~2610
     try
     {
       t2 = x ? NULL : ctor(&obj2,x+1)~2620
        foo( x  ? t1 : t2);
      }
      finally
      {
         if (x) dtor(&obj1); ~2630
      }
   }
   finally
   {
     if (!x) dtor(&obj2); ~2640
   }
}
```

FIG. 27

```
   _x              =    ENTER proc
   t140            =    CMP(NE) _x, 0
                        CBRANCH(NE) t140, $L4, $L5 ~2710
   $L4:
      t134         =    CALL ctor, &objl, _x; $PROPAGATE
2720  t135         =    ASSIGN t134                       2721
   -> tv141-       =    ASSIGN [t135)
      $t142        =    ASSIGN 1
                        GOTO $L6
   $L5:
      t137         =    ADD _x, 1                        2731
2730  t138         =    CALL ctor, &obj2, t137; $PROPAGATE
   -> t139         =    ASSIGN t138
      tv141-       =    ASSIGN [t139)
      $t142        =    ASSIGN 0
                        GOTO $L6
   $L6:
      t145         =    ASSIGN tv141-
                        CALL bar, t145
                        FINAL $OBJ1, $L11
   $L11:
                        FINAL $OBJ2, $L12
   $OBJ1:
      r1           =  ┌->FINALLY
      t144         =  │  CMP(EQ) $t142, 0
                      │  CBRANCH(EQ) t144, $L9, $L10
   $L9:               │
                      │  CALL dtor, &objl  $PROPAGATE
                      │  GOTO $L10
   $L10:      2740 ~  │
                      └->ENDFINALLY; rl, [$L11), $PROPAGATE
   $OBJ2:
      r2           =  ┌->FINALLY
      t143         =  │  CMP(EQ) $t142, 1
                      │  CBRANCH(EQ) t143, $L7, $L8
   $L7:               │
                      │  CALL dtor, &obj2  $PROPAGATE
                      │  GOTO $L8
   $L8:       2750 ~  │
                      └->ENDFINALLY; r2, [$L12), $PROPAGATE
   $PROPAGATE:
                        UNWIND
                        EXIT
   $L12:
                        EXIT
```

FIG. 28

```
Obj foo(int x)
 {
  Obj a , b;~2810
  bar();
  if (x == 0)
  {
    return Obj(1);~2820
  }
  else
  {
    return Obj(2);~2830
  }
 }
```

FIG. 29A

```
Obj foo(int x)
    {
     CALL ctor (&a); $unwind;
     CALL ctor (&b); $final_a;
     bar(); $final_b;
     if (x == 0)
     {
       CALL ctor Obj (&r1,1) ; $final_b;~2960
       $f1 = 1;~2950
       FINAL $final_b1; L2
       L2:
         FINAL $final_a1, L1
       L1:
         $f1 = 0;
         FINAL $final_r1, Lret
       Lret:
         return r1;~2910
     }
     CALL ctor Obj(&r2, 2); $final_b;
     $f2 = 1;
     FINAL $final_b2, L3
     L3:
       FINAL $final_a2, L4
     L4:
       $f2 = 0;
       FINAL $final_r2, Lret2
     Lret2:
       return r2;
```

FIG. 29B

```
$final_b:
        e, R = FINALLY
        DTOR (&b); $final_a;
        ENDFINALLY e, R, $final_a;
      $final_a:
        e, R = FINALLY
        CALL DTOR (&a); $unwind;
        ENDFINALLY e, R, HANDLER:$unwind;
      $final_b1:
        e, R = FINALLY
        CALL DTOR (&b); $final_a1; ~2930
        ENDFINALLY e, R, [L2], $final_a1;
      $final_a1:
        e, R = FINALLY
        CALL DTOR (&a); $final_r1; ~2920
        ENDFINALLY e, R, [L1]; $final_r1;
      $final_b2:
        e, R = FINALLY
        CALL DTOR (&b); $final_a2;
        ENDFINALLY e, R,[L3]; $final_a2;
      $final_a2:
        e, R = FINALLY
        CALL DTOR (&a); $final_r2;
        ENDFINALLY e, R, [L4]; $final_r2;
      $final_r1:
        e, R = FINALLY
        if ($f1 == 1) CALL DTOR (&r1);$unwind; ~2940
        ENDFINALLY e, R, [Lret1]; $unwind;
     $final_r2:
        e, R = FINALLY
        if ($f2 == 1) CALL DTOR (&r2); $unwind;
        ENDFINALLY e, R,[Lret2]; $unwind;
     }
```

FIG. 30

```
void proc()
{
    class1 obj1; // S1 - create an obj of type Class1
    obj1.foo();  // S2 - calling a method on obj.1
    throw foo;   // S3
}
```

FIG. 31

```
void proc()
{
    class1 obj1;                                        // S1
    class1 temp;
    try {
      obj1.foo();                    3110               // S2
          temp.copy_ctor(obj1);//
          special_throw(&temp, &dtor_of_class1)         // S3
    }                                                       3120
    finally {
          dtor_of_class1(obj1);
    }                               3130
}
```

FIG. 32

```
    ENTER proc
    CALL ctor1(&obj1); $PROPAGATE
    CALL foo(&obj1);   $DTOR1
    CALL copy_ctor(&temp, &obj1); $DTOR1
    FINAL $DTOR1;
    THROWVAL &temp, &dtor_of_class1; $PROPAGATE
$DTOR1:               \3220              \3230               ↖
    e2, r2 = FINALLY                                          3210
    CALL DTOR1(&obj1); $PROPAGATE
    ENDFINALLY e1, r2, $PROPAGATE, $END
$PROPAGATE:
    UNWIND
    EXIT;
$END:
    EXIT;
```

FIG. 33

```
void proc()
{
    __try {
        foo();
    } __except(filter()) {
        body();
    }
    next();
}
```

FIG. 34

```
    ENTER proc
$LABEL:
    SEHENTER;    $HANDLER~3410
    CALL foo();  $HANDLER~3420
    GOTO $NEXT;
$HANDLER:~3430
    x = FILTER
    t = CALL filter();~3450
    ENDRESUMEFILTER t, $HANDLERBODY, $END, $LABEL
$HANDLERBODY:                                    \3440
    CALL body(); $PROPAGATE
    GOTO $NEXT;
$NEXT:
    CALL next();
    EXIT;
$END:
    EXIT;
```

FIG. 35

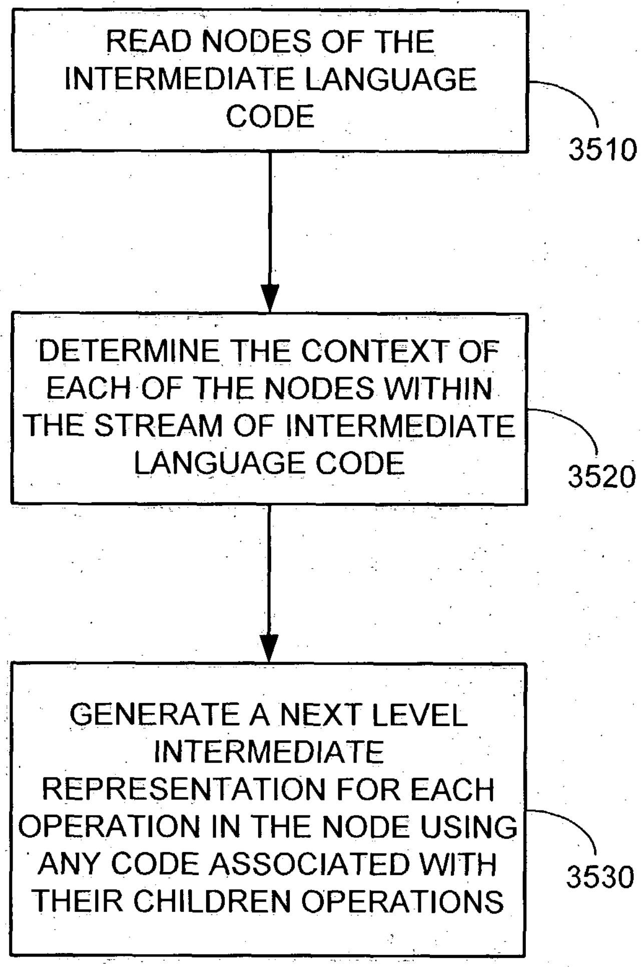

READ NODES OF THE INTERMEDIATE LANGUAGE CODE
3510
DETERMINE THE CONTEXT OF EACH OF THE NODES WITHIN THE STREAM OF INTERMEDIATE LANGUAGE CODE
3520
GENERATE A NEXT LEVEL INTERMEDIATE REPRESENTATION FOR EACH OPERATION IN THE NODE USING ANY CODE ASSOCIATED WITH THEIR CHILDREN OPERATIONS
3530

FIG. 36

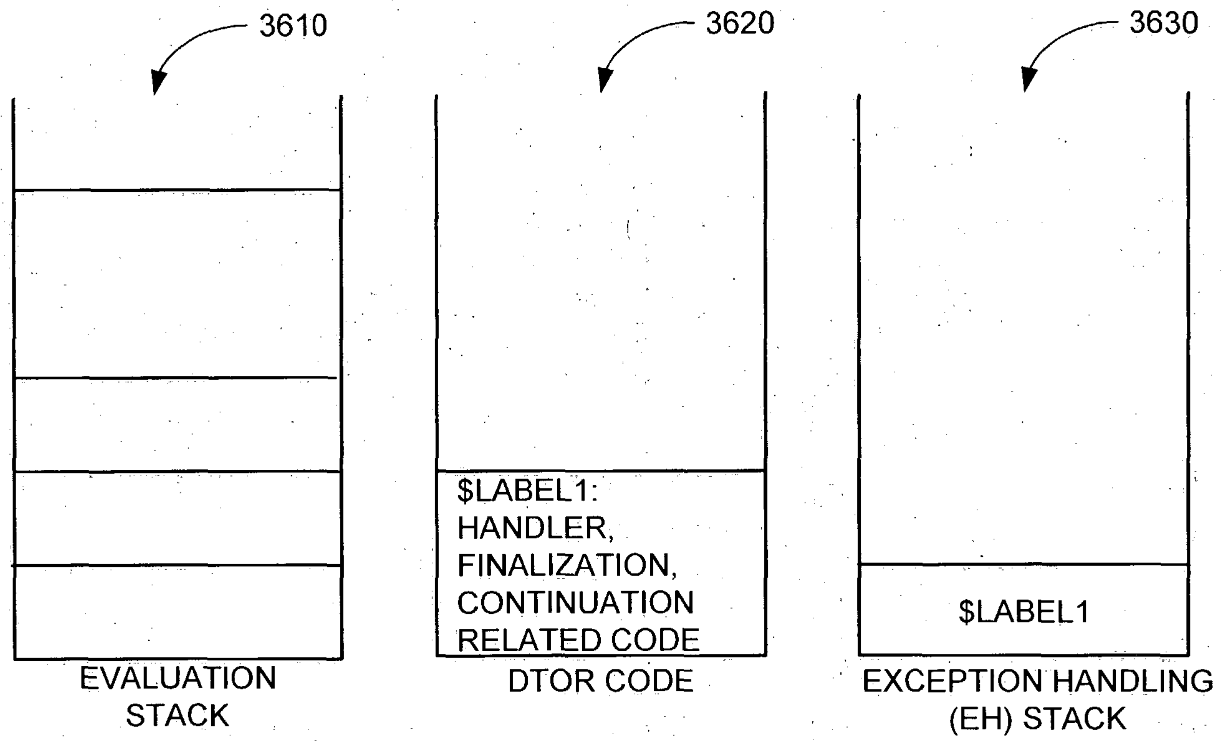
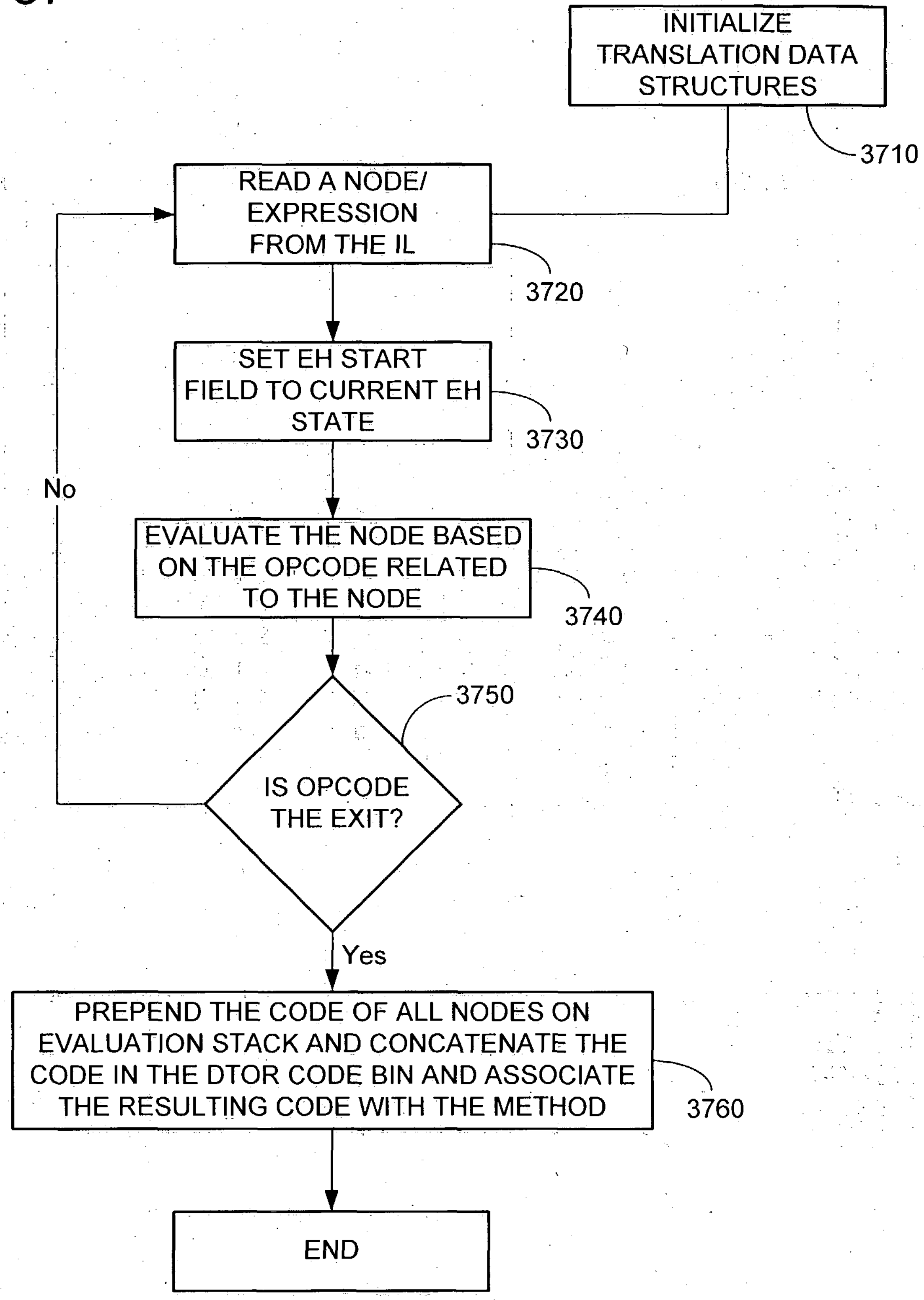

3610
3620
3630
$LABEL1:
HANDLER,
FINALIZATION,
CONTINUATION
RELATED CODE
$LABEL1
EVALUATION STACK
DTOR CODE
EXCEPTION HANDLING (EH) STACK

FIG. 37


INITIALIZE TRANSLATION DATA STRUCTURES
3710
READ A NODE/ EXPRESSION FROM THE IL
3720
SET EH START FIELD TO CURRENT EH STATE
3730
EVALUATE THE NODE BASED ON THE OPCODE RELATED TO THE NODE
3740
IS OPCODE THE EXIT?
3750
No
Yes
PREPEND THE CODE OF ALL NODES ON EVALUATION STACK AND CONCATENATE THE CODE IN THE DTOR CODE BIN AND ASSOCIATE THE RESULTING CODE WITH THE METHOD
3760
END

Gregory L. Maurer, Klarquist Sparkman, LLP, 121 SW Salmon St., Suite 1600, Portland, Oregon 97204, (503) 226-7391; Inventor: Grover et al.; Title: AN INTERMEDIATE REPRESENTATION FOR MULTIPLE EXCEPTION HANDLING MODELS; Attorney Docket No.: 3382-65591; Express Mail Label No. EV339203824US; Date of Deposit: June 26, 2003; Page 31 of 34

FIG. 38A

```
>>> IL for function ?proc@@YAXXZ:
OPpragma pragma(31)     PR_PRAGSTAT Pragma Status: 0x00800000
OPpragma pragma(32)     PR_INLINE Inline Status: 0x0010
OPpragma pragma(2)      PR_FILENAME Filename key(0x20)
OPpragma pragma(1)      PR_LINENUMBER Line(22)
OPpragma pragma(35)     PR_WARNING 10:OFF; 16:ERR; 72:ERR; 86:OFF;
93:OFF; 120:OFF; 205:OFF; 206:OFF; 217:OFF; 228:OFF; 231:OFF; 246:OFF;
OPblock
OPblock
OPname(?proc@@YAXXZ) symbol(0x288)
OPentry
OPeolist
***** function entry says proc has no parameters ********

**** ctor call for class1 *******************************
OPblock
OPpragma pragma(1)      PR_LINENUMBER Line(23)
OPname(??0class1@@QAE@XZ) symbol(0x25f)              3810
OPname(obj1) symbol(0x28a)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction address size(4) align(3) Const functype(20)
OPeolist
******** ***********************************


*** dtor call for class1 *******************************
OPname(??1class1@@QAE@XZ) symbol(0x260)
OPname(obj1) symbol(0x28a)                           3820
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction void size(0) align(1) functype(20)
OPeolist
*******************************************************
```

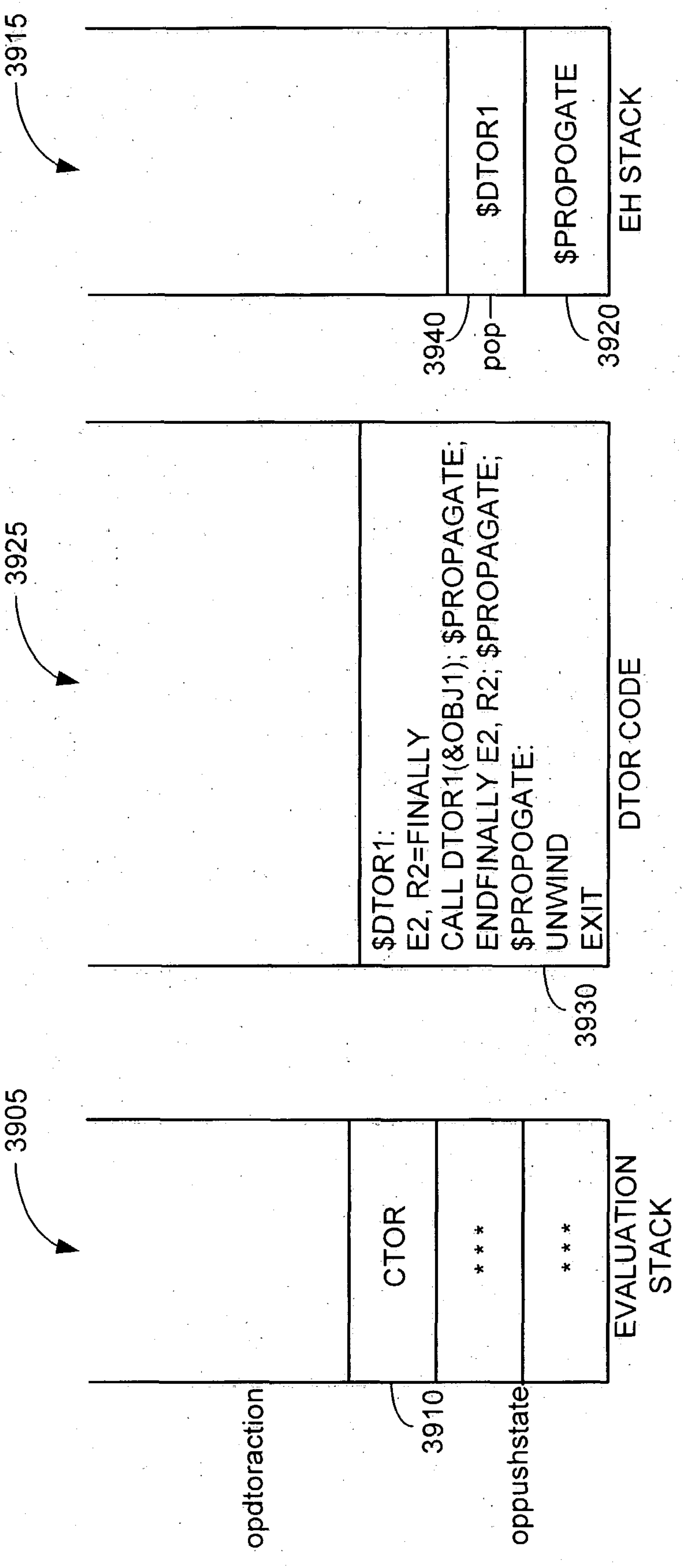
FIG. 38B

```
OPpushstate address size(4) align(3) EH Flags 0x00000011
                                                          3830
OPexpression
OPpragma pragma(1)      PR_LINENUMBER Line(24)
OPname(?foo@class1@@QAEXXZ) symbol(0x261)
OPname(obj1) symbol(0x28a)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction void size(0) align(1) functype(20)
OPeolist
OPexpression
OPpragma pragma(1)      PR_LINENUMBER Line(25)
OPname(??0class2@@QAE@XZ) symbol(0x274)
OPname(obj2) symbol(0x28b)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction address size(4) align(3) Const functype(20)
OPeolist
OPname(??1class2@@QAE@XZ) symbol(0x275)
OPname(obj2) symbol(0x28b)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction void size(0) align(1) functype(20)
OPeolist
```

FIG. 38C

```
OPpushstate address size(4) align(3) EH Flags 0x00000011
OPexpression
OPpragma pragma(1)      PR_LINENUMBER Line(26)
OPname(?bar@class2@@QAEXXZ) symbol(0x276)
OPname(obj2) symbol(0x28b)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3)
OPconstant integer size(4) align(3) constant(0|0x0)
OPfield address size(4) align(3) Const
OPextract address size(4) align(3) Const
OPmfunc address size(4) align(3)
OPfunction void size(0) align(1) functype(20)
OPeolist
OPexpression
OPpragma pragma(1)      PR_LINENUMBER Line(27)
OPdtoraction cnt(2) EH Flags 0x00000031
OPexpression                                  3840
OPgoto symbol(0x289)
OPendblock icon(2)
OPlabel symbol(0x289)
OPexit
OPendblock icon(1)
OPendblock icon(0)
```

FIG. 39


3905
CTOR
* * *
* * *
EVALUATION STACK
3910
opdtoraction
oppushstate
3925
$DTOR1:
E2, R2=FINALLY
CALL DTOR1(&OBJ1); $PROPAGATE;
ENDFINALLY E2, R2; $PROPAGATE;
$PROPOGATE:
UNWIND
EXIT
DTOR CODE
3930
3915
$DTOR1
$PROPOGATE
EH STACK
3940
pop
3920